(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 November 2006 (30.11.2006)

PCT

(10) International Publication Number
WO 2006/127546 A2

(51) International Patent Classification:
G06T 5/00 (2006.01) *H04N 7/30* (2006.01)

(21) International Application Number: PCT/US2006/019641

(22) International Filing Date: 19 May 2006 (19.05.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/683,240	20 May 2005 (20.05.2005)	US
11/437,515	18 May 2006 (18.05.2006)	US

(71) Applicant *(for all designated States except US)*: **NTT DOCOMO, INC.** [US/US]; 181 Metro Drive, Suite 300, San Jose, CA 95110 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **LELESCU, Dan** [CA/US]; 18325 Serra Avenue, Morgan Hill, CA 95037 (US). **BOSSEN, Frank** [NL/CH]; 71 Ch. Des Petits-bois, CH-1228 Plan-ies Quates XX (CH).

(74) Agents: **MALLIE, Michael, J.** et al.; BLAKELY SOKOLOFF TAYLOR & ZAFMAN LLP, 12400 Wilshire Boulevard, 7th Floor, Los Angeles, CA 90025 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RU, SC, SD, SE, SG, SK, SL, SM, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, YU, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

(54) Title: METHOD AND APPARATUS FOR NOISE FILTERING IN VIDEO CODING

(57) Abstract: A method and apparatus is disclosed herein for encoding and/or decoding video frame data, hi one embodiment, the video coder comprises a noise filtering module to operate on transformed frame data and perform signal estimation on a plurality of transform coefficients by estimating signal power for each of the plurality of transform coefficients, comparing the signal power of said each coefficient with at least one threshold, and setting the value of said each transform coefficient based, at least in part, on results of comparing the signal power to the at least one threshold.

WO 2006/127546 A2

METHOD AND APPARATUS FOR NOISE FILTERING IN VIDEO CODING

PRIORITY

[0001] The present patent application claims priority to and incorporates by reference the corresponding provisional patent application serial no. 60/683,240, titled, "Method And Apparatus For Noise Filtering in Video Coding," filed on May 20, 2005.

FIELD OF THE INVENTION

[0002] The present invention relates to the field of video coding; more particularly, the present invention relates to performing noise filtering in video coding based on estimation of the signal power of coefficients and a comparison of those estimates to one or more theresholds.

BACKGROUND OF THE INVENTION

[0003] Conventional methods of hybrid video coding either encode a video frame by itself, without reference to other frames in the sequence, or encode a video frame by predicting it from other, already coded and reconstructed, frames in the sequence. This process routinely operates at block-level in the images. In either case, the frames that are reconstructed following the coding process contain distortions introduced by this process, which can be modeled as noise. This noise may manifest itself visibly as coding artifacts, such as blocking and ringing artifacts. Some of the reconstructed frames stored in a frame store typically used by a video coder and decoder are further used as reference for predicting other frames in the sequence while others are not. In the previous case, the quality of the reconstructed frame stored in the frame store is not only important for its display but, through its use in the prediction process, influences the quality of subsequently coded frames which rely on it for reference purposes. In the latter case, for frames that are not further used as reference, their quality is important for display purposes. Thus, in either case, a procedure to remove the coding noise is beneficial for the quality of the reconstructed video sequence and an increase in the objective coding efficiency.

[0004] Most modern codecs include some type of filtering to reduce the level of resultant coding noise. The filtering can operate outside or inside of the coding loop in a codec. For loop filters which operate inside the coding loop, generally the same operation is performed in the encoder and in the decoder using the same available data, such that no side information specific to this filtering needs to be transmitted explicitly from the coder to the decoder. Noise filtering methods commonly used include simple low-pass filtering of the reconstructed frame data to smooth out the images and decrease the visibility of coding artifacts, and loop filters that adapt their strength based on local image characteristics (such as block edges).

[0005] Out-of-coding-loop filtering techniques, such as a lowpass filtering of a reconstructed (noisy) frame at the decoder are usually not sufficient for improving the overall image quality of the decoded video sequence. See, H. C. Reeve and J. S. Lim, "Reduction of Blocking Effects in Image Coding," Opt. Eng., Vol. 23, no. 1, pp.34-37. Jan/Feb. 1984. In-loop filters that perform filtering inside the coding and decoding loop in an encoder and decoder, respectively, have superior performance. For more information, see Wiegand, et al., "Overview of the H.264/AVC Video Coding Standard," IEEE Transactions on Circuits and Systems for Video Technology, on page(s): 560-576, vol. 13, Issue: 7, July 2003. They usually adapt the filtering strength in a heuristic manner, based on the quantization regime used by the video codec and the characteristics of the signal being filtered. Also, since they operate in the coding loop, they cause an improvement in the quality of the reference frames used for prediction, thus improving the efficiency of the coding process.

[0006] In contrast to operating heuristically, there exist filtering techniques that are based on signal estimation from noise using a particular signal model. For more information, see A. Papoulis, "Probability, Random Variables, and Stochastic Processes", 3rd edition, New York, McGraw-Hill, 1991.

[0007] Overcomplete denoising techniques that operate in the transform domain provide additionally the advantage of determining multiple reconstructed instances (estimates) for the same sample position in a frame, which can then be combined (e.g., averaged) in order to improve the estimation quality. See, R. R. Coifman and D. L. Donoho, "Translation Invariant Denoising," in Wavelets and Statistics, Springer Lecture Notes in Statistics 103, pp. 125-150, New York, Springer

Verlag. Existing denoising approaches, which attempt to extract the signal from coding noise, have at their core a signal estimator. Since usually the statistics of the problem are not known exactly, it is necessary to make some assumptions about the unknowns and choose a strategy for estimating the signal. One of the most popular strategies is to optimize the signal estimation for the worst-possible choice of unknowns, resulting in robust estimators. For more information, see Y. C. Eldar, A. B.-Tal, and A. Nemirovski, "Linear Minimax Regret Estimation of Deterministic Parameters with Bounded Data Uncertainties," IEEE Transactions on Signal Processing, Vol. 52, No. 8, Aug. 2004. It is well-known that under this design constraint, the estimators that result tend to be too conservative because the performance is optimized for the worst-case scenario.

[0008] The performance of prior methods of coding noise filtering is limited by some intrinsic characteristics of these methods and the assumptions made about the nature of the noise. Simple low-pass filtering techniques that are applied to reduce coding artifacts are not effective in handling the diversity of visual information in video frames, and tend to have widely-varying performance for these sequences (lack of performance control). Adaptive, in-loop boundary filtering in video coding uses heuristics that do not guarantee optimal solutions in some well-defined sense. Also, existing denoising techniques based on the use of a signal model, determine estimators under unrealistic assumptions about the nature of the signal and noise, in particular by assuming that the signal and noise are uncorrelated. This is not the case for the signal and coding noise encountered in image and video coding. The performance of these filtering techniques suffers, since they are poorly matched to the existing problem conditions.

SUMMARY OF THE INVENTION

[0009] A method and apparatus is disclosed herein for encoding and/or decoding video frame data. In one embodiment, the video coder comprises a noise filtering module to operate on transformed frame data and perform signal estimation on a plurality of transform coefficients by estimating signal power for each of the plurality of transform coefficients, comparing the signal power of said each coefficient with at least one threshold, and setting the value of said each transform coefficient based, at least in part, on results of comparing the signal power to the at least one threshold.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] The present invention will be understood more fully from the detailed description given below and from the accompanying drawings of various embodiments of the invention, which, however, should not be taken to limit the invention to the specific embodiments, but are for explanation and understanding only.

[0011] **Figure 1** is a block diagram of one embodiment of a video coder.

[0012] **Figure 2** is a flow diagram of one embodiment of a video coding process.

[0013] **Figure 3** is a block diagram of one embodiment of a noise filter module.

[0014] **Figure 4** is a flow diagram of one embodiment of a noise filtering process.

[0015] **Figure 5** is a flow diagram of another embodiment of a noise filtering process.

[0016] **Figure 6** is a flow diagram of yet another embodiment of a noise filtering process.

[0017] **Figure 7** is a flow diagram of one embodiment of an estimation process.

[0018] **Figure 8** is a flow diagram of another embodiment an estimation process.

[0019] **Figure 9** is a block diagram of one embodiment of a video decoder.

[0020] **Figure 10** is a flow diagram of one embodiment of a video decoding process.

[0021] **Figure 11** is a block diagram of one embodiment of a computer system.

DETAILED DESCRIPTION OF THE PRESENT INVENTION

[0022] Apparatus and methods for implementing a signal denoising technique for video coding and decoding are disclosed. In one embodiment, a filtering apparatus operates on the reconstructed frame data generated by a video codec, and the filtering process takes place in an overcomplete transform domain corresponding to the frame data. An estimation of the signal from the noise incurred by the coding process is performed in this transform domain. In one embodiment, this estimation process comprises estimating the signal power corresponding to a transform coefficient, comparing the estimated signal power to one or more thresholds, and adjusting the transform coefficient based on the results of the comparison. Thus, the estimation process uses the estimated signal power, noise power, and the values of the threshold(s).

[0023] In one embodiment, given the overcomplete representation, for each data sample position in a frame being processed there correspond multiple estimates of the sample at that position, as they are produced by the estimation process. These initial estimates of each data sample are further filtered in order to obtain the final estimate for a data sample at a given position in the processed video frame. Such filtering may comprise averaging.

[0024] Thus, adaptive estimation in the overcomplete transform domain is used to extract the signal from noise incurred during the coding process. In one embodiment, the estimator in the noise filter to estimate the signal power operates based on assumptions about the signals and noise in the coding process. Also, in one embodiment, the signal estimators are designed to achieve a balance between robustness and performance such as to avoid the drawbacks of related art estimators described.

[0025] The use of this signal denoising technique results in improved objective rate-distortion performance, as well as better subjective (visual) quality of the decoded video frames compared to related art techniques for video frame filtering.

[0026] In the following description, numerous details are set forth to provide a more thorough explanation of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form, rather than in detail, in order to avoid obscuring the present invention.

[0027] Some portions of the detailed descriptions which follow are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers, or the like.

[0028] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the following discussion, it is appreciated that throughout the description, discussions utilizing terms such as "processing" or "computing" or "calculating" or "determining" or "displaying" or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data represented as physical (electronic) quantities within the computer system's registers and memories into other data similarly represented as physical quantities within the computer system memories or registers or other such information storage, transmission or display devices.

[0029] The present invention also relates to apparatus for performing the operations herein. This apparatus may be specially constructed for the required purposes, or it may comprise a general purpose computer selectively activated or reconfigured by a computer program stored in the computer. Such a computer program may be stored in a computer readable storage medium, such as, but is not limited to, any type of disk including floppy disks, optical disks, CD-ROMs, and magnetic-optical disks, read-only memories (ROMs), random access memories

(RAMs), EPROMs, EEPROMs, magnetic or optical cards, or any type of media suitable for storing electronic instructions, and each coupled to a computer system bus.

[0030] The algorithms and displays presented herein are not inherently related to any particular computer or other apparatus. Various general purpose systems may be used with programs in accordance with the teachings herein, or it may prove convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these systems will appear from the description below. In addition, the present invention is not described with reference to any particular programming language. It will be appreciated that a variety of programming languages may be used to implement the teachings of the invention as described herein.

[0031] A machine-readable medium includes any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory ("ROM"); random access memory ("RAM"); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.); etc.

Overview of Video Coder Embodiments

[0032] In one embodiment, a video codec contains a filtering module that processes frame data in order to reduce or eliminate the coding noise, and thus improves both the objective and subjective quality of the coding process. The noise filter employs adaptive estimation techniques operating in a domain constructed by a representation of the frame data in an overcomplete basis to extract the video signal from the noisy frame data. The apparatus and methods described herein perform estimation and filtering operations in the overcomplete basis representation of the frame data in a different manner than in existing related art, and their use results in superior performance when compared to the related art. Some differences from related art include, but are not limited to, the structure of the estimator used for extracting the signal from noise, the estimation of other variables that are used by this estimator, and by the structure of the filter that is used to process the signal estimates produced by the estimator in order to determine the final filtered data samples that are used either for display, or for both reference and display purposes as described above.

[0033] The use of the estimation techniques and/or the filtering processes described herein invention results in an increased data coding efficiency reflected in a better rate-distortion profile of the coding process, as well as in the improved visual quality of the decoded video sequence through the reduction or elimination of coding artifacts, compared to related art methods.

[0034] Figure 1 is a block diagram of one embodiment of a video coder. Referring to Figure 1, the video coder comprises a motion estimation module (MEM) 129, a motion-compensated prediction module (MCPM) 131, a memory 123, a transform coding module (TCM) 114, a motion data processing module (MDPM) 127, a noise filter module (NFM) 125, a frame store (FS) (e.g., memory) 126, and switches 118, 124, 128, 141, and 142. Transform coder 114 in turn includes a transform module 111, a quantizer module 112, and an entropy coding module 113. The frame data at the input of the transform coder 114 may consist of video frames, or displaced frame difference (DFD) frames. A DFD frame is obtained in the video coder by taking the difference between data in a video frame and its prediction generated at the output of MCPM 131. The difference is output from subtractor 103 that receives video frame 101 and an I frame or intra-block (when switch 141 is closed). MCPM 131 generates a prediction based on data consisting of motion-compensated reconstructed video frames stored in frame store 126. The motion compensation takes place using motion information generated by the motion estimation module 129. This process is enabled by a configuration of the switches in Figure 1 that will be discussed below. Alternatively, when a video coder (VC) directly encodes a video frame corresponding to the intra-coding mode of conventional coders, without the use of prediction, MEM 129, MCPM 131, and MDPM 127 are not utilized.

[0035] The video coder typically operates sequentially on block-partitioned frame data, which consists of luminance and chrominance values at each position in a block.

[0036] Figure 2 is a flow diagram of one embodiment of a video coding process. The process is performed by processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both.

[0037] In Figure 2, blocks from a frame are read and processed sequentially as described by the following process, until all blocks of the current frame have been processed. Referring to Figure 2, the process begins by processing logic setting an index variable i, used to index the blocks in the current frame, equal to 1 (processing block 201). Next, processing logic reads block c(i) from the current frame (processing block 202).

[0038] In a first phase of the video coder operation, switches in Figure 1 are configured as follows. Switches 141, 142, and 128 are open or closed depending on the type of blocks being processed (open for intra-coded blocks, and closed for predicted blocks as in a conventional video process). Switch 124 is open.

[0039] Processing logic also initializes index k equal to 1, where the index k is a position from the search area corresponding to the current block c(i) (processing block 203) and retrieves a reference block b(k) in the reference frame corresponding to the position k in the motion estimation search area (processing block 204). A motion estimation process is performed, whereby candidate positions within a search area in a reference frame from the frame store are sequentially searched to identify a best-match reference block for the current block c(i). A goodness of match measure is used to quantify the differences between block c(i) and b(k). The motion estimation process attempts to find a block b(k) in a search area in reference frame so as to minimize a measure of difference d(i,k) between the current block c(i) and the reference block b(k). This position is specified by a motion vector mv(i,k) originating at the current block c(i) and pointing to the reference block b(k) in the reference frame.

[0040] Processing logic computes a measure d(i,k) of the difference between blocks c(i) and b(k) and stores that in memory (processing block 205). In one embodiment, the difference d(i,k) is quantified for example by a sum of absolute differences between the samples of c(i) and b(k). Thus, for current block c(i), a position k inside the search area is tested by retrieving the corresponding reference block b(k) and computing d(i,k) and the error d(i,k) is stored in memory.

[0041] Processing logic then increments the variable k by 1 (processing block 206) and tests whether the variable k is less than or equal to K, which is the total number of positions in the search area (processing block 207). If all the search area positions from the reference frame corresponding to the current block c(i) have not been examined (i.e., k<K), processing logic transitions back to processing block 204 and the process continues from there. If they have (i.e., k is not less than K), processing logic transitions to processing block 208 where processing logic finds the best match b(n) in the sense of the minimum d(i,k) between c(i) and b(k). In one embodiment, after all K search area positions from a reference frame corresponding to the current block c(i) have been visited, processing logic of MEM 129 finds the smallest error d_{min} among the array of values d(i,k) stored in memory, which corresponds to a specific reference block denoted by b_{min} whose coordinates are specified by motion vector mv(i).

[0042] Thereafter, processing logic stores the motion vector mv(i,n) in memory (processing block 209) and decides, and stores, the coding mode m(i) for block c(i), where the coding mode m(i) for block c(i) is INTRA or predicted (processing block 210). In one embodiment, processing logic of MEM 129 determines the coding mode cmode(i) of the current block c(i) by comparing a cost of encoding the current block c(i) when intra coding its samples with a cost of encoding block c(i) predictively with respect to the block b_{min}. If the first cost is smaller, the mode of the block c(i) is marked as INTRA; otherwise the block is marked as a predicted block.

[0043] If the coding mode for block c(i) is not INTRA, then the motion vector mv(i) is stored in memory. The mode cmode(i) of the current block is also written in memory, and the process moves on to the next block c(i) in the current frame by having processing logic increments the index variable i by 1 (processing block 211) and testing whether i is less than N, which is total number of blocks in the current frame (processing block 212). If the index i is less than N, processing logic transitions to processing block 202 and the process continues from there. If not, processing logic transitions to processing block 213 where the index variable i is set equal to 1.

[0044] Once all the blocks in the current frame have been processed as described above, the motion compensation process is initiated. The motion compensation process begins by processing logic reading the original current block c(i) from the current frame (processing block 214). In one embodiment, the blocks are read sequentially. Processing logic tests whether the coding mode for block c(i) is equal to INTRA (processing block 215). In one embodiment, this is performed by processing logic in MCPM 131. If the coding mode for block c(i) is equal to INTRA, processing logic transitions to processing block 219. If not, processing logic performs motion compensation using mv(i,n) to fetch the block predictor b(i) from the reference frame (processing block 216), computes the prediction error e(i) according to the formula:

$$e(i) = c(i)-p(i)$$

, where the prediction block p(i) is produced by MCPM 131 using motion vector mv(i) stored in memory for block c(i) (processing block 217), entropy codes the motion information (processing block 218), and transitions to processing block 219. In one embodiment, the motion vector data corresponding to the current block is sent to MDPM 127 where it is processed (e.g., DPCM encoded with respect to neighboring blocks' motion vectors), and sent to the entropy coder module 113 for encoding.

[0045] At processing block 219, processing logic entropy codes the coding mode of c(i). Processing logic then transforms code the block data c(i) or e(i) depending on the mode m(i) (processing block 220), reconstructs the current block (processing block 221), and writes that to memory (e.g., memory 123) (processing block 222).

[0046] The process described above is repeated for all the blocks in the current frame. To that end, processing logic then increments the index i by one (processing block 223) and tests whether the index i is less than N, which is the total number of blocks in the current frame (processing block 224). If it is, processing transitions to processing block 214 and the process continues from there. If not, processing logic filters the noisy frame data in memory (e.g., memory 123) (processing block 225) and writes the filtered data to the frame store (e.g., frame store 126) (processing block 226). Thereafter, the process ends.

[0047] After all the blocks in the current frame have been processed, switch 124 is then closed. The reconstructed frame data stored in memory 123 is processed by noise filter module 125 as described below. Noise filter module 125 stores the data it processes in frame store 126.

Noise Filtering

[0048] Figure 3 is a block diagram of one embodiment of a noise filtering module (e.g., noise filtering module 125 of Figure 1). Referring to Figure 3, the noise filtering module comprises a memory 301, a spatial mask position generation module (MPGM)) 302, a transform module (TM) 303, an estimator module (EM) 304, a memory 305, an inverse transform module (ITM) 306, a memory 307, and a filter module (FM) 308. Memory 301 provides a mask structure 310 to mask position generator 302. In response thereto, mask position generator 302 sets forth positions in the frame 311 and provides these positions to transform module 303, inverse transform 306 and filter 308. Transform module 303 applies a transform to decoded frame data 312 to generate transform coefficients 313 of samples under the mask set forth by mask position generator 302. Estimator 305 receives transform coefficients 313 and generates denoised transform coefficients 314 in response thereto. Denoised transformed coefficients 314 are stored in memory 305. Subsequently, inverse transform module 306 applies an inverse transform to denoised transform coefficients, based on the mask from mask position generator 302, to generate denoised samples 315 under the mask. These are stored in memory 307. Filter module 308 performs filtering on denoised samples 315, based on the mask from mask position generator 302, to produce reconstructed frame samples 316.

[0049] The input to the noise filter module consists of the reconstructed frame data 312 (e.g., decoded frame data) stored in the memory module 123. The noise filter module outputs the filtered frame data 316 (e.g., reconstructed frame samples) to frame store 126. The operation of the noise filter module proceeds as illustrated by the flow diagrams depicted in Figures 4, 5, and 6.

[0050] Figure 4 is a flow diagram of embodiment of a noise filtering process. The process is performed by processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of estimator 304 of the noise filter module of Figure 3.

[0051] The process performed by the noise filter module begins by retrieving from memory (e.g., 123) the reconstructed frame data stored therein. Using the reconstructed frame data, processing logic computes a set of variables V (processing block 401). This set of auxiliary variables V will be further used in the estimation process. Based on the quantization regime used by the codec, the estimator computes a measure of the quantization noise introduced by the coding process. For purposes herein, the quantization step used in the encoding process is denoted by q. In one embodiment, processing logic computes the auxiliary variable

$$v^2 = \alpha\frac{q^2}{4},$$

where α is a constant, and v^2 represents an estimate of the noise power. Processing logic in the estimator also determines the value of a variable b, based on the characteristics of coding regime used in the encoder. In this case, b is computed as $b = \beta v^2$, where β is a constant. In another embodiment, processing logic computes additional auxiliary variables as follows:

$$t_1 = \mu v^2 - (\mu - 1)b$$

$$x_1 = \sqrt{b}$$

$$x_2 = \sqrt{t_1}$$

$$z_1 = \frac{1}{x_2 - x_1}$$

$$z_2 = x_1 z_1,$$

where μ is a selectable constant.

[0052] In yet another embodiment, processing logic computes the following auxiliary variables:

$$t_1 = \mu v^2 - (\mu - 1)b$$

$$x_1 = \sqrt{b}$$

$$x_2 = \sqrt{t_1}$$

$$y_2 = \frac{\mu}{\mu+1} x_2,$$

$$h_1 = \frac{y_2}{x_2 - x_1},$$

$$h_2 = \frac{y_2 x_1}{x_2 - x_1},$$

where μ is a selectable constant.

[0053] Processing logic then sets mask S with a region of support R (processing block 402). For example the mask size (region of support R) can be made equal to that of a block used in the encoding process.

[0054] Thereafter, processing logic initializes (i,j) to the position of a block in the frame (processing block 403). This block position (i,j) is situated on the grid of blocks customarily used by a video codec to code/decode a frame is selected in the frame currently being filtered from memory (e.g., memory 123).

[0055] Processing logic also initializes (m,n) to a position around and including (i,j) (processing block 404). The position (m,n) represents the position where the mask S is shifted compared to (i,j). Processing logic then moves the mask S to position (m,n) in the frame (processing block 405).

[0056] Processing logic applies the transform T (e.g., a Discrete Cosine transform (DCT)) to samples under mask S (processing block 406) and denoises the transformed coefficients using processing logic in the estimator (processing block 407). The N transform coefficients corresponding to the data samples covered by the current position of the mask S are denoted herein by s(k), k=1...N. The resulting transform coefficients are processed by processing logic in the estimator module. Different embodiments for this operation are illustrated in the flow diagrams of Figures 7 and 8, which are described in greater detail below.

[0057] In the flow diagram of Figure 7, processing logic in the estimator module retrieves a transform coefficient s(k) and estimates the signal power by computing the value $s^2(k)$. Processing logic then compares the signal power with a threshold and determines the estimator value a(k) corresponding to a coefficient s(k) as follows:

```
if $s^2(k) <= b$
    $a(k) = 0;$
else
    $a(k) = \frac{s^2(k) - b}{s^2(k) + v^2 - 2b}$.
```

[0058] Processing logic in the estimator then determines the processed transform coefficient p(k) according to:

$$p(k) = a(k)s(k).$$

[0059] In an alternate embodiment, using the process depicted in the flow diagram of Figure 8, processing logic in the estimator makes use of the auxiliary variables V previously defined above and computes the value of the estimator for each coefficient s(k) as follows:

```
if $s^2(k) <= b$
    $a(k) = 0;$
else if $b < s^2(k) <= t_1$
    $a(k) = z_1 s(k) - z_2$
else
    $a(k) = 1.$
```

[0060] Processing logic in the estimator then determines the processed transform coefficient p(k) as:

$$p(k) = a(k)s(k).$$

[0061] In an alternate embodiment, also using the process depicted in the flow diagram of Fig. 8, for each coefficient *s*(*k*), processing logic in the estimator computes the value of processed coefficient p(k) as follows:

```
if $s^2(k) <= b$
    $p(k) = 0;$
else if $b < s^2(k) <= t_1$
    $p(k) = h_1 s(k) - h_2;$
else
    $p(k) = s(k)$. Figure 8 will be described in more detail below.
```

[0062] Processing logic stores the coefficient p(k) in memory (e.g., memory 305), irrespective of the whether the process of Figures 7 or 8 are used. If not all coefficients have been processed the process described above is repeated.

[0063] Next, processing logic applies the inverse transform T^{-1} (e.g., an inverse DCT transform) to the processed transform coefficients p(k) under mask S (processing block 408), resulting in samples r(k) corresponding to the current position (m,n) of the mask S in the frame. Processing logic stores these processed samples under mask S in memory (e.g., memory 307) (processing block 409).

[0064] Processing logic then sets (m,n) to the next position (processing block 410) and tests whether all positions (m,n) in the region of support R have been visited (processing block 411). If not all the positions of the mask S around block position (i,j) have been processed, processing transitions to processing block 405, the next shift position (m,n) of mask S relative to (i,j) is selected and the process described above is repeated.

[0065] If all the positions of the mask S around block position (i,j) have been processed, processing block transitions to processing block 412, where processing logic selects the next block position (i,j) on the block grid in the frame tests whether all block positions (i,j) of the block being processed in the frame have been processed (processing block 413). If not, processing logic transitions to processing block 404 where the process continues from there. If so, processing transitions to processing block 414. In this way, the process above is repeated until all block positions (i,j) have been visited.

[0066] At processing block 414, processing logic filters the reconstruction versions corresponding to each sample position in the frame using a filter (e.g., filter 308) (processing block 414) and writes the resulting filtered data samples f(x,y) for the current frame being processed into the frame store (e.g., frame store 126) (processing block 415). In one embodiment, processing logic filters the processed samples stored in memory (e.g., memory 307) corresponding to each sample in the current frame. In one embodiment, this filtering is performed by filter 308 of the noise filter module 125. Thus, for each position (x,y) in the current frame being processed by the noise filter module 125, there are multiple processed and reconstructed samples $r^l(x,y)$ stored in memory 307 by the estimator 304, where l=1...L indexes the positions that the mask S can occupy around a block grid position in the frame, and where at each such location the mask S region of support R contains sample position (x,y). In one embodiment, the filter performs a weighted average operation on its input data $r^l(x,y)$ corresponding to a frame position (x,y):

$$f(x,y) = \sum_l w_l r^l(x,y) \quad ,$$

where w_l are weights selected based on statistics generated by processing logic in the estimator. In one embodiment, the weights are all selected to be equal, $w_l = \frac{1}{L}$, and correspond to an averaging operation. In an alternative embodiment, the weights are selected to be proportional to the number of zero-valued processed coefficients p(k,) resulting after the application of the estimator at a position l of the mask S.

[0067] Then the process ends.

[0068] Figure 5 is a flow diagram of an alternative embodiment of a noise filtering process performed by the noise filtering module. The process is performed by processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both. In one embodiment, the processing logic is part of estimator 304 of the noise filter module of Figure 3.

[0069] Referring to Figure 5, the process begins by processing logic generating a mask S with a region of support R (processing block 501). For example the mask size (region of support R) can be made equal to that of a block used in the encoding process.

[0070] Processing logic computes a set of variables V(k) (processing block 502). Since a transform (e.g., DCT) is applied to the samples under mask S (at processing block 303), the transform coefficients can be ranked as customary for such transforms. This rank is denoted by k. In this embodiment, the variables v(k) depend on the rank k of the transform coefficients generated under mask S. This set of auxiliary variables v(k) will be further used in the estimation process. Based on the quantization regime used by the codec, processing logic computes a measure of the quantization noise introduced by the coding process. For purposes herein, the quantization step used in the encoding process is denoted by q. In this case, processing logic computes a variable

$$v^2(k) = \alpha(k)\frac{q^2}{4}.$$

where, α can be adapted based on various factors such as the rank k of the coefficient s(k) inside the mask S, in which $\alpha = \alpha\ (k)$ case. For example, α is increased for higher rank coefficients. If v is a function of k, then b also becomes a function of coefficient rank k, $b(k) = \beta v^2(k)$. In another embodiment, processing logic computes additional auxiliary variables depending on the rank k of transform coefficients under to mask S (position within S), as follows:

$$t_1(k) = \mu v^2(k) - (\mu - 1)b(k)$$

$$x_1(k) = \sqrt{b(k)}$$

$$x_2(k) = \sqrt{t_1(k)}$$

$$z_1(k) = \frac{1}{x_2(k) - x_1(k)}$$

$$z_2(k) = x_1(k)z_1(k),$$

where μ is a selectable constant.

[0071] Alternatively, in another embodiment, processing logic computes the following auxiliary variables:

$$t_1(k) = \mu v^2(k) - (\mu - 1)b(k)$$

$$x_1(k) = \sqrt{b(k)}$$

$$x_2(k) = \sqrt{t_1(k)}$$

$$y_2(k) = \frac{\mu}{\mu + 1}x_2(k),$$

$$h_1(k) = \frac{y_2(k)}{x_2(k) - x_1(k)},$$

$$h_2(k) = \frac{y_2(k)x_1(k)}{x_2(k) - x_1(k)},$$

where μ is a selectable constant.

[0072] Processing logic then initializes (i,j) to the position of a block in the frame (processing block 503). In one embodiment, a block position (i,j) situated on the grid of blocks customarily used by a video codec to code/decode a frame is selected in the frame currently being filtered from memory (e.g., memory 123).

[0073] Processing logic also initializes (m,n) to a position around and including (i,j) (processing block 504). The position (m,n) is the position where the mask S is shifted compared to (i,j). Processing logic moves the mask S to position (m,n) in the frame (processing block 505).

[0074] Processing logic applies the transform T (e.g., DCT) to the samples under mask S (processing block 506) and denoises the transform coefficients (processing block 507). The N transform coefficients corresponding to the data samples covered by the current position of the mask S are denoted herein by s(k), k=1...N. The resulting transform coefficients are processed by processing logic in the estimator module. Different embodiments for this operation are illustrated in the flow diagrams of Figures 7 and 8, which are described in greater detail below.

[0075] In the flow diagram of Figure 7, processing logic in the estimator module retrieves a transform coefficient s(k) and estimates the signal power by computing the value $s^2(k)$. Processing logic then compares the signal power with a threshold and determines the estimator value for a coefficient s(k) as follows:

```
if $s^2(k)$ <=b(k)
        a(k) = 0;
else
```

$$a(k) = \frac{s^2(k) - b(k)}{s^2(k) + v^2(k) - 2b(k)}.$$

[0076] Processing logic in the estimator then determines the processed transform coefficient p(k) according to:

$$p(k) = a(k)s(k).$$

[0077] In an alternate embodiment, using the process depicted in the flow diagram of Figure 8, processing logic in the estimator makes use of the auxiliary variables and computes the value of the estimator for each coefficient s(k) as follows:

```
if $s^2(k)$<=b(k)
```

```
        a(k) = 0;
else if b(k)< $s^2(k)$<= $t_1$(k)
        $a(k) = z_t(k)s(k) - z_2(k)$
else
        a(k) = 1.
```

[0078] Processing logic in the estimator then determines the processed transform coefficient p(k) as:

$$p(k) = a(k)s(k).$$

[0079] In an alternate embodiment, also using the process depicted in the flow diagram of Fig. 8, for each coefficient $s(k)$, processing logic in the estimator computes the value of processed coefficient p(k) as follows:

```
if  $s^2(k)$ <=b(k)
        p(k) = 0;
else if b(k)< $s^2(k)$<= $t_1$(k)
        $p(k) = h_1(k)s(k) - h_2(k)$;
else
        p(k) = s(k).
```

[0080] Figure 8 will be described in more detail below.

[0081] Processing logic stores the coefficient p(k) (e.g., in memory 305), irrespective of the whether the process of Figures 7 or 8 are used. If not all coefficients have been processed the process described above is repeated.

[0082] After denoising, processing logic applies an inverse transform T^{-1} (e.g., an inverse DCT) to the transform coefficients under mask S (processing block 508), resulting in samples r(k) corresponding to the current position (m,n) of the mask S in the frame, and stores the reconstructed samples under mask S in memory (e.g., memory 307) (processing block 509).

[0083] Processing logic then sets (m,n) to the next position (processing block 510) and tests whether all positions (m,n) in the region of support R have been visited (processing block 511). If not all the positions of the mask S around block position (i,j) have been processed, processing transitions to processing block 505, the next position (m,n) of mask S relative to (i,j) is selected and the process described above is repeated.

[0084] If all the positions of the mask S around block position (i,j) have been processed, processing block transitions to processing block 512, where processing logic selects the next block position (i,j) on the block grid in the frame tests whether all block positions (i,j) of the block being processed in the frame have been processed (processing block 513). If not, processing logic transitions to processing block 504 where the process continues from there. If so, processing transitions to processing block 514. In this way, the process above is repeated until all block positions (i,j) have been visited.

[0085] At processing block 514, processing logic filters the reconstructed versions corresponding to each sample position in the frame using a filter (processing block 514). In one embodiment, the filter used is filter 308. Thus, for each position (x,y) in the current frame being processed by the noise filter module 125, there are multiple processed and reconstructed samples $r^l(x,y)$ stored in memory 307 by the estimator, where l=1...L indexes the positions that the mask S can occupy around a block grid position in the frame, and where at each such location its region of support R contains sample position (x,y). In one embodiment, the filter 308 performs a weighted average operation on its input data $r^l(x,y)$ corresponding to a frame position (x,y):

$$f(x,y)=\sum_l w_l r^l(x,y),$$

where w_l are weights selected based on statistics generated by the estimator. In one embodiments, the weights are all be selected to be equal, $w_1 = \frac{1}{L}$. In an alternative embodiment, the weights are selected to be proportional to the number of zero-valued processed coefficients p(k) resulting after the application of the estimator at a position l of the mask S.

[0086] Processing logic writes resulting filtered data samples f(x,y) for the current frame being processed to the frame store (e.g., frame store 126) (processing block 515).

[0087] Thereafter, the process ends.

[0088] Figure 6 is a flow diagram of yet another alternative embodiment of a noise filtering process performed by the noise filtering module. The process is performed by processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both.

[0089] Referring to Figure 6, the process begins by processing logic generating mask S with a region of support R (processing block 601). In one embodiment, the mask size (region of support R) can be made equal to that of a block used in the encoding process.

[0090] Processing logic then initializes (i,j) to the position of a block in the frame (processing block 602). In one embodiment, a block position (i,j) situated on the grid of blocks customarily used by a video codec to code/decode a frame is selected in the frame currently being filtered from memory (e.g., memory 123).

[0091] Processing logic also initializes (m,n) to a position around and including (i,j) (processing block 603). The position (m,n) is the position where the mask S is shifted compared to (i,j). Processing logic moves the mask S to position (m,n) in the frame (processing block 604).

[0092] Processing logic in the estimator computes a set of variables V(m,n,k) that depend not only on the rank k of the transform coefficients generated by 303 at the current mask position (processing block 605) but also on the position (m,n) of the mask S in the frame. The set of auxiliary variables that will be further used in the estimation process. Based on the quantization regime used by the codec, processing logic computes a measure of the quantization noise introduced by the coding process. For purposes herein, the quantization step used in the encoding process is denoted by q. In this case, processing logic computes a variable

$$v^2(m,n,k) = \alpha(m,n,k)\frac{q^2}{4}.$$

where α can be adapted based on various factors such as the rank k of the coefficient s(k) inside the mask S, and the position (m,n) of mask S around (i,j) in which case $\alpha = \alpha(m,n,k)$. For example, α is increased for higher rank coefficients. If v is a function of (m,n,k) then b also becomes a function of coefficient rank m,n,k, $b(m,n,k) = \beta v^2(m,n,k)$. In another embodiment, processing logic computes additional auxiliary variables depending on the rank m,n,k of transform coefficients mapped to mask S (position within S), as follows:

$$t_1(m,n,k) = \mu v^2(m,n,k) - (\mu - 1)b(m,n,k)$$

$$x_1(m,n,k) = \sqrt{b(m,n,k)}$$

$$x_2(m,n,k) = \sqrt{t_1(m,n,k)}$$

$$z_1(m,n,k) = \frac{1}{x_2(m,n,k) - x_1(m,n,k)}$$

$$z_2(m,n,k) = x_1(m,n,k)z_1(m,n,k),$$

where μ is a selectable constant.

[0093] Alternatively, in another embodiment, processing logic computes the following auxiliary variables:

$$t_1(m,n,k) = \mu v^2(m,n,k) - (\mu - 1)b(m,n,k)$$

$$x_1(m,n,k) = \sqrt{b(m,n,k)}$$

$$x_2(m,n,k) = \sqrt{t_1(m,n,k)}$$

$$y_2(m,n,k) = \frac{\mu}{\mu+1} x_2(m,n,k),$$

$$h_1(m,n,k) = \frac{y_2(m,n,k)}{x_2(m,n,k) - x_1(m,n,k)},$$

$$h_2(m,n,k) = \frac{y_2(m,n,k)x_1(m,n,k)}{x_2(m,n,k) - x_1(m,n,k)},$$

where μ is a selectable constant.

[0094] Processing logic applies the transform T (e.g., DCT) to samples under mask S (processing block 606) and denoises the transformed coefficients using processing logic in the estimator (processing block 607). The N transform coefficients corresponding to the data samples covered by the current position of the mask S are denoted herein by s(m,n,k), k=1...N. The resulting transform coefficients are processed by processing logic in the estimator module. Different embodiments for this operation are illustrated in the flow diagrams of Figures 7 and 8, which are described in greater detail below.

[0095] In the flow diagram of Figure 7, processing logic in the estimator module retrieves a transform coefficient s(m,n,k) and estimates the signal power by computing the value $s^2(m,n,k)$. Processing logic then compares the signal power with a threshold and determines the estimator value for a coefficient s(m,n,k) as follows:

```
if $s^2(m,n,k) <= b(m,n,k)$
    $a(m,n,k) = 0$;
else
    $a(m,n,k) = \frac{s^2(m,n,k) - b(m,n,k)}{s^2(m,n,k) + v^2(m,n,k) - 2b(m,n,k)}$.
```

[0096] Processing logic in the estimator then determines the processed transform coefficient p(m,n,k) according to:

$$p(m,n,k) = a(m,n,k)s(m,n,k).$$

[0097] In an alternate embodiment, using the process depicted in the flow diagram of Figure 8, processing logic in the estimator makes use of the auxiliary variables and computes the value of the estimator for each coefficient s(m,n,k) as follows:

```
if $s^2(m,n,k) <= b(m,n,k)$
    $a(m,n,k) = 0$;
else if $b(m,n,k) < s^2(m,n,k) <= t_1(m,n,k)$
    $a(m,n,k) = z_1(m,n,k)s(m,n,k) - z_2(m,n,k)$
else
    $a(m,n,k) = 1$.
```

[0098] Processing logic in the estimator then determines the processed transform coefficient p(m,n,k) as:

$$p(m,n,k) = a(m,n,k)s(m,n,k).$$

[0099] In an alternate embodiment, also using the process depicted in the flow diagram of Fig. 8, for each coefficient $s(m,n,k)$, processing logic in the estimator computes the value of processed coefficient p(m,n,k) as follows:

```
if $s^2(m,n,k) <= b(m,n,k)$
    $p(m,n,k) = 0$;
else if $b(m,n,k) < s^2(m,n,k) <= t_1(m,n,k)$
    $p(m,n,k) = h_1(m,n,k)s(m,n,k) - h_2(m,n,k)$;
else
```

p(m,n,k) = s(m,n,k).

[00100] Figure 8 will be described in more detail below.

[00101] Processing logic stores the coefficient p(k), irrespective of the whether the process of Figures 7 or 8 are used, is stored in memory (e.g., memory 305). If not all coefficients have been processed the process described above is repeated.

[00102] Next, processing logic applies the inverse transform T^{-1} to the transform coefficients under mask S (processing block 608), resulting in samples r(m,n,k) corresponding to the current position (m,n) of the mask S in the frame. Processing logic stores these reconstruction samples under mask S in memory (e.g., memory 307) (processing block 609).

[00103] Processing logic then sets (m,n) to the next position (processing block 610) and tests whether all positions (m,n) in the region of support R have been visited (processing block 611). If not all the positions of the mask S around block position (i,j) have been processed, processing transitions to processing block 605, the next shift position (m,n) of mask S relative to (i,j) is selected and the process described above is repeated.

[00104] If all the positions of the mask S around block position (i,j) have been processed, processing block transitions to processing block 612, where processing logic selects the next block position (i,j) on the block grid in the frame tests whether all block positions (i,j) of the block being processed in the frame have been processed (processing block 613). If not, processing logic transitions to processing block 604 where the process continues from there. If so, processing transitions to processing block 614. In this way, the process above is repeated until all block positions (i,j) have been visited.

[00105] At processing block 614, processing logic filters the reconstruction versions corresponding to each sample position in the frame using a filter (e.g., filter 308) (processing block 614) and writes the resulting filtered data samples f(x,y) for the current frame being processed into the frame store (e.g., frame store 126) (processing block 615). In one embodiment, processing logic filters the estimated samples stored in memory (e.g., memory 307) corresponding to each sample in the current frame. In one embodiment, this filtering is performed by filter 308 of the

noise filter module 125. Thus, for each position (x,y) in the current frame being processed by the noise filter module 125, there are multiple processed and reconstructed samples $r^l(x,y)$ stored in memory by the estimator 304, where l=1...L indexes the positions that the mask S can occupy around a block grid position in the frame, and where at each such location its region of support R contains sample position (x,y). In one embodiment, the filter performs a weighted average operation on its input data $r^l(x,y)$ corresponding to a frame position (x,y):

$$f(x,y) = \sum_l w_l r^l(x,y) ,$$

where w_l are weights selected based on statistics generated by processing logic in the estimator. In one embodiment, the weights are all be selected to be equal, $w_1 = \frac{1}{L}$. In an alternative embodiment, the weights are selected to be proportional to the number of zero-valued processed coefficients p(k,) resulting after the application of the estimator at a position l of the mask S.

[00106] Then the process ends.

<u>Examples of Estimation Processes</u>

[00107] Figure 7 is a flow diagram of one embodiment of an estimation process. The process is performed by processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both.

[00108] Referring to Figure 7, the process begins by processing logic retrieving a transform coefficient (processing block 701) and estimating the signal power of the transform coefficient (processing block 702). In one embodiment, the signal power is estimated by taking the square of the transform coefficient.

[00109] The processing logic then tests whether the signal power is less than a threshold (processing block 703). One embodiment of the threshold has been described above in paragraph 0052 above. If it is, processing logic sets the coefficient to 0 (processing block 704) and processing transitions to processing block 706 where the coefficient is stored. If not, processing logic adjusts the value of the coefficient based on signal power, noise power and threshold by executing the logic described above (processing block 705) and processing transitions to processing block 706 where the coefficient is stored.

[00110] After storing the coefficient, processing logic tests whether all coefficients have been processed (processing block 707). If not, processing logic transitions to processing block 701 and the process continues from that point. If so, the process ends.

[00111] Figure 8 is a flow diagram of an alternative embodiment of an estimation process. The process is performed by processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both.

[00112] Referring to Figure 8, the process begins by processing logic retrieving a transform coefficient (processing block 801) and estimating the signal power of the transform coefficient (processing block 802). In one embodiment, the signal power is estimated by taking the square of the transform coefficients. The processing logic then tests whether the signal power is less than a threshold (processing block 803). One embodiment of the threshold is described above in paragraph 0071. If it is, processing logic sets the coefficient to 0 (processing block 804) and processing transitions to processing block 806 where the coefficient is stored. If not, processing logic tests whether the signal power is less than a second threshold (processing block 805). The embodiment of the threshold is described above in paragraph 0093. If it is, processing logic adjusts the value of the coefficient based on the signal power, noise power and threshold by executing the logic described above (processing block 806) and processing transitions to processing block 808 where the coefficient is stored. If not, processing logic leaves the coefficient unchanged (processing block 807) and transitions to processing block 808 where the coefficient is stored.

[00113] After storing the coefficient, processing logic tests whether all coefficients have been processed (processing block 809). If not, processing logic transitions to processing block 701 and the process continues from that point. If so, the process ends.

An Example of a Video Decoder

[00114] Figure 9 is a block diagram of one embodiment of a video decoder (VD). Referring to Figure 9, the video decoder comprises a transform decoder 913, adder 904, a memory module 905, a motion compensation and prediction module (MCPM) 907, a motion data processing module (MDPM) 908, a frame store 909, a noise filter module (NFM) 906, and switches 941-943. In one embodiment, transform decoder 913 includes an entropy decoder 901, an inverse quantizer 902, and an inverse transform module 903.

[00115] The video decoder reconstructs the video frame data according to the process specified below in conjunction with the flow diagram of Figure 10. Switches 942 and 943 are open or closed depending on the decoded block type (intra-coded or predicted) similar to the operation of conventional coding processes. Switch 941 is initially open.

[00116] Figure 10 is a flow diagram of one embodiment of video decoding process. The process is performed by processing logic that may comprise hardware (circuitry, dedicated logic, etc.), software (such as is run on a general purpose computer system or a dedicated machine), or a combination of both.

[00117] Referring to Figure 10, the process begins by processing logic initializing an index variable i equal to 1 (processing block 1001).

[00118] Processing logic then decodes the coding mode for the current block c(i) in the video frame being decoded (processing block 1002) and tests whether the coding mode of block c(i) is equal to INTRA (processing block 1003). If it is, processing logic intra-decodes the current block c(i) according to a conventional process (processing block 1004) and transitions to processing block 1009. If not, processing logic decodes the block prediction error e(i) for the current block c(i) using transform decoder 913 (processing block 1005), decodes motion vector mv(i) of the current block c(i) using the transform decoder 913 and MDPM 908 (processing block 1006), motion compensates the current block c(i) using mv(i) and the reference frame in the frame store to determine a block predictor u(i) (processing block 1007). In one embodiment, the decoded motion vector mv(i) is used by MCPM 908 to fetch a reference block (predictor) u(i) from a reference frame stored in frame store 909.

[00119] Next, processing logic reconstructs the current block according to the following equation:

$$rc(i)=u(i)+e(i)$$

(processing block 1008). In other words, the current block rc(i) is reconstructed by adding the prediction error e(i) to the predictor samples u(i).

[00120] Thereafter, processing logic writes the reconstructed data in memory (e.g., memory 905) (processing block 1009). That is, the data corresponding to the reconstructed block rc(i) is written into memory. The processing logic then moves on to the next block position in the current frame being decoded by incrementing i by one (processing block 1010).

[00121] Processing logic tests whether the index variable i is less than N which is the total number of blocks in the current frame being processed (processing block 1011). If it is, processing logic transitions to processing block 1002 where the process is continues from that point. In this way, the process described above is repeated for all blocks in the current frame being decoded. If not, switch 941 is closed and processing logic filters the frame data in memory (e.g., memory 905) using a noise filter module (e.g., noise filter module 906) (processing logic 1012). In one embodiment, the decoded frame data stored in memory module 905 is read by the loop filtering module which executes the same process as the one described for the loop filter module 125 in the video encoder. Then processing logic writes the filtered data to the frame store (e.g., frame store 909) (processing block 1013).

[00122] Thereafter, the process ends.

[00123] Thus, the methods and apparatus described herein are used to determine a video frame data filtering that results in superior objective and subjective (image quality) performance compared to related art solutions.

An Example Computer System

[00124] Figure 11 is a block diagram of an exemplary computer system that may perform one or more of the operations described herein. Referring to Figure 11, computer system 1100 may comprise an exemplary client or server computer system. Computer system 1100 comprises a communication mechanism or bus 1111 for communicating information, and a processor 1112 coupled with bus 1111 for

processing information. Processor 1112 includes a microprocessor, but is not limited to a microprocessor, such as, for example, Pentium™, PowerPC™, Alpha™, etc.

[00125] System 1100 further comprises a random access memory (RAM), or other dynamic storage device 1104 (referred to as main memory) coupled to bus 1111 for storing information and instructions to be executed by processor 1112. Main memory 1104 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 1112.

[00126] Computer system 1100 also comprises a read only memory (ROM) and/or other static storage device 1106 coupled to bus 1111 for storing static information and instructions for processor 1112, and a data storage device 1107, such as a magnetic disk or optical disk and its corresponding disk drive. Data storage device 1107 is coupled to bus 1111 for storing information and instructions.

[00127] Computer system 1100 may further be coupled to a display device 1121, such as a cathode ray tube (CRT) or liquid crystal display (LCD), coupled to bus 1111 for displaying information to a computer user. An alphanumeric input device 1122, including alphanumeric and other keys, may also be coupled to bus 1111 for communicating information and command selections to processor 1112. An additional user input device is cursor control 1123, such as a mouse, trackball, trackpad, stylus, or cursor direction keys, coupled to bus 1111 for communicating direction information and command selections to processor 1112, and for controlling cursor movement on display 1121.

[00128] Another device that may be coupled to bus 1111 is hard copy device 1124, which may be used for marking information on a medium such as paper, film, or similar types of media. Another device that may be coupled to bus 1111 is a wired/wireless communication capability 1125 to communication to a phone or handheld palm device.

[00129] Note that any or all of the components of system 1100 and associated hardware may be used in the present invention. However, it can be appreciated that other configurations of the computer system may include some or all of the devices.

[00130] Whereas many alterations and modifications of the present invention will no doubt become apparent to a person of ordinary skill in the art after having read the foregoing description, it is to be understood that any particular embodiment shown and described by way of illustration is in no way intended to be considered limiting. Therefore, references to details of various embodiments are not intended to limit the scope of the claims which in themselves recite only those features regarded as essential to the invention.

CLAIMS

We claim:

1. A video coder for encoding the data in a video frame, the video coder comprising:

a noise filtering module to operate on transformed frame data and perform signal estimation on a plurality of transform coefficients by

estimating signal power for each of the plurality of transform coefficients,

comparing the signal power of said each coefficient with at least one threshold, and

setting the value of said each transform coefficient based, at least in part, on results of comparing the signal power to the at least one threshold.

2. A method comprising:

estimating signal power for each of the plurality of transform coefficients;

comparing the signal power of said each coefficient with at least one threshold; and

setting the value of said each transform coefficient based, at least in part, on results of comparing the signal power to the at least one threshold.

3. An article of manufacture having one or more computer readable medium storing instructions thereon which, when executed by a system, cause the system to perform a method comprising:

estimating signal power for each of the plurality of transform coefficients;

comparing the signal power of said each coefficient with at least one threshold; and

setting the value of said each transform coefficient based, at least in part, on results of comparing the signal power to the at least one threshold.

4. A method for denoising video frame data comprising:

retrieving frame data from a memory;

applying a transform to samples in the frame under a mask to create transform coefficients;

denoising the transform coefficients;

applying an inverse transform to the denoised transform coefficients; and

filtering the denoised frame samples corresponding to a position in a frame to obtain the filtered data sample at that position.

5. An article of manufacture having one or more computer readable medium storing instructions thereon which, when executed by a system, cause the system to perform a method for denoising video frame data comprising:

retrieving frame data from a memory;

applying a transform to samples in the frame to create transform coefficients;

denoising the transform coefficients;

applying an inverse transform to the denoised transform coefficients; and

filtering the denoised frame samples corresponding to a position in a frame to obtain the filtered data sample at that position.

6. A noise filtering module comprising:

a transform module to transform frame spatial data to another domain suitable for signal processing;

an estimator module to generate denoised estimates of transform coefficients corresponding to frame data by estimating signal power, comparing the estimated signal power to one or more thresholds, and adjusting the transform coefficients according to results of the comparison and the values of the noise power and thresholds;

an inverse transform module to apply an inverse transform to the denoised transform coefficients and reverting the data under the mask to its original domain;

a filter module to obtain a filtered sample at each frame position by combining its estimates resulting from processing by the estimator.

7. An estimator module comprising:

means for estimating signal power;

means for comparing the estimated signal power to one or more thresholds; and

means for adjusting signal transform coefficient values according to the results of comparing the signal transform coefficient values with one or more thresholds and based on the values of the estimated signal power, noise power and the one or more thresholds.

8. A video decoder for decoding the data in a video frame, the video decoder comprising:

a noise filtering module to operate on transformed frame data and perform signal estimation on a plurality of transform coefficients by

estimating signal power for each of the plurality of transform coefficients,

comparing the signal power of said each coefficient with at least one threshold, and

setting the value of said each transform coefficient based, at least in part, on results of comparing the signal power to the at least one threshold.

9. A decoding method comprising:

decoding a prediction error for a block of frame data;

decoding a motion vector associated with the block;

performing motion compensation on the block using the motion vector and a reference frame to determine a block predictor;

reconstructing the block using the block predictor and the prediction error; and

filtering reconstructed block samples by generating a plurality of transform coefficients from the frame data, comparing an estimate of signal power of the transform coefficients to one or more thresholds to generate denoised transform coefficients in response to the transform coefficients, and filtering denoised samples to create reconstructed frame samples.

10. An article of manufacture having one or more computer readable medium storing instructions thereon which, when executed by a system, cause the system to perform a decoding method comprising:

decoding a prediction error for a block of frame data;

decoding a motion vector associated with the block;

performing motion compensation on the block using the motion vector and a reference frame to determine a block predictor;

reconstructing the block using the block predictor and the prediction error; and

filtering reconstructed block samples by generating a plurality of transform coefficients from the frame data, comparing an estimate of signal power of the transform coefficients to one or more thresholds to generate denoised transform coefficients in response to the transform coefficients, and filtering denoised samples to create reconstructed frame samples.

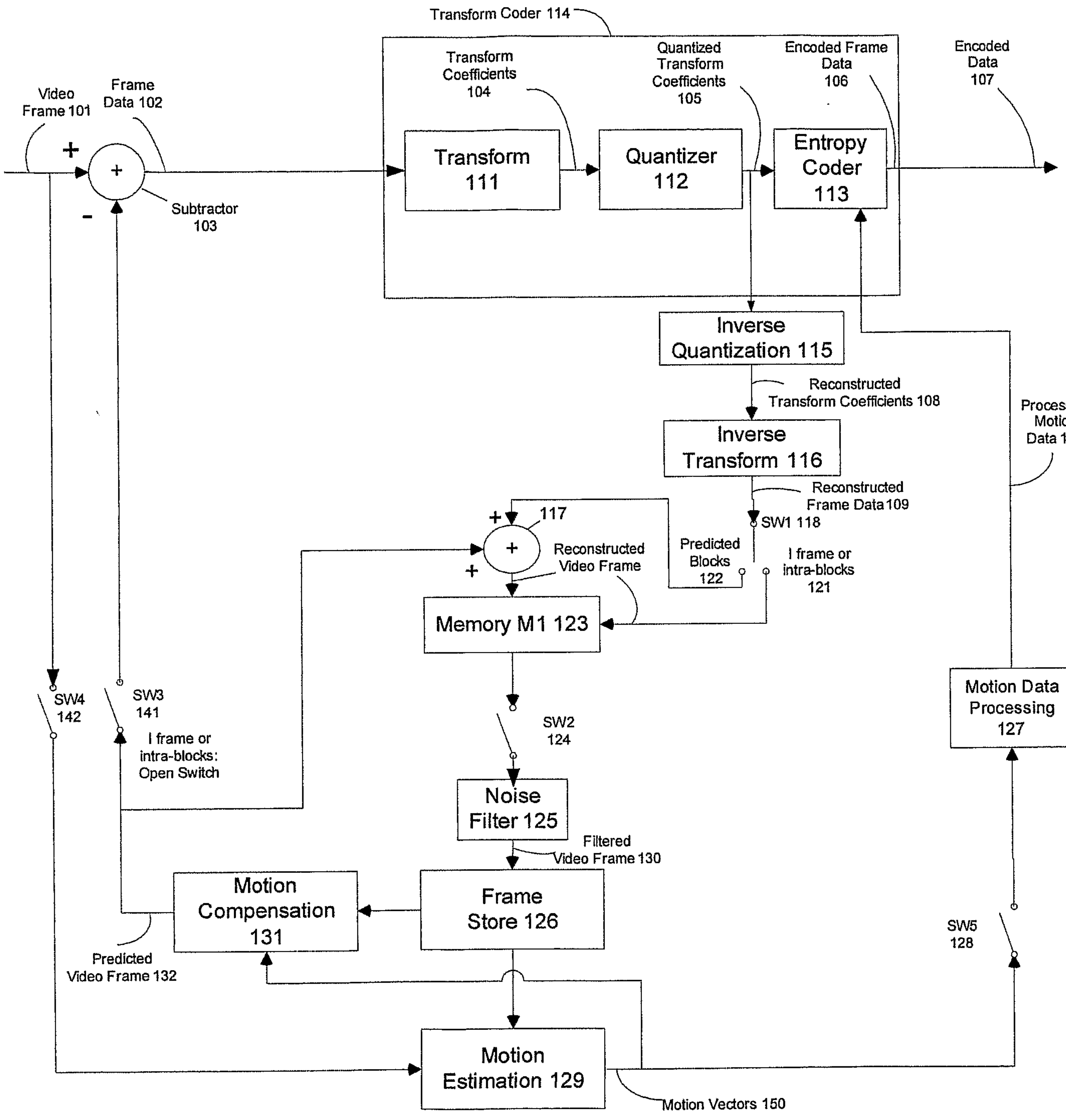
Transform Coder 114
Transform Coefficients 104
Quantized Transform Coefficients 105
Encoded Frame Data 106
Encoded Data 107
Video Frame 101
Frame Data 102
+
-
Subtractor 103
Transform 111
Quantizer 112
Entropy Coder 113
Inverse Quantization 115
Reconstructed Transform Coefficients 108
Inverse Transform 116
Reconstructed Frame Data 109
SW1 118
Processed Motion Data 110
117
Reconstructed Video Frame
Predicted Blocks 122
I frame or intra-blocks 121
Memory M1 123
SW4 142
SW3 141
I frame or intra-blocks: Open Switch
SW2 124
Motion Data Processing 127
Noise Filter 125
Filtered Video Frame 130
Motion Compensation 131
Frame Store 126
SW5 128
Predicted Video Frame 132
Motion Estimation 129
Motion Vectors 150

FIG. 1

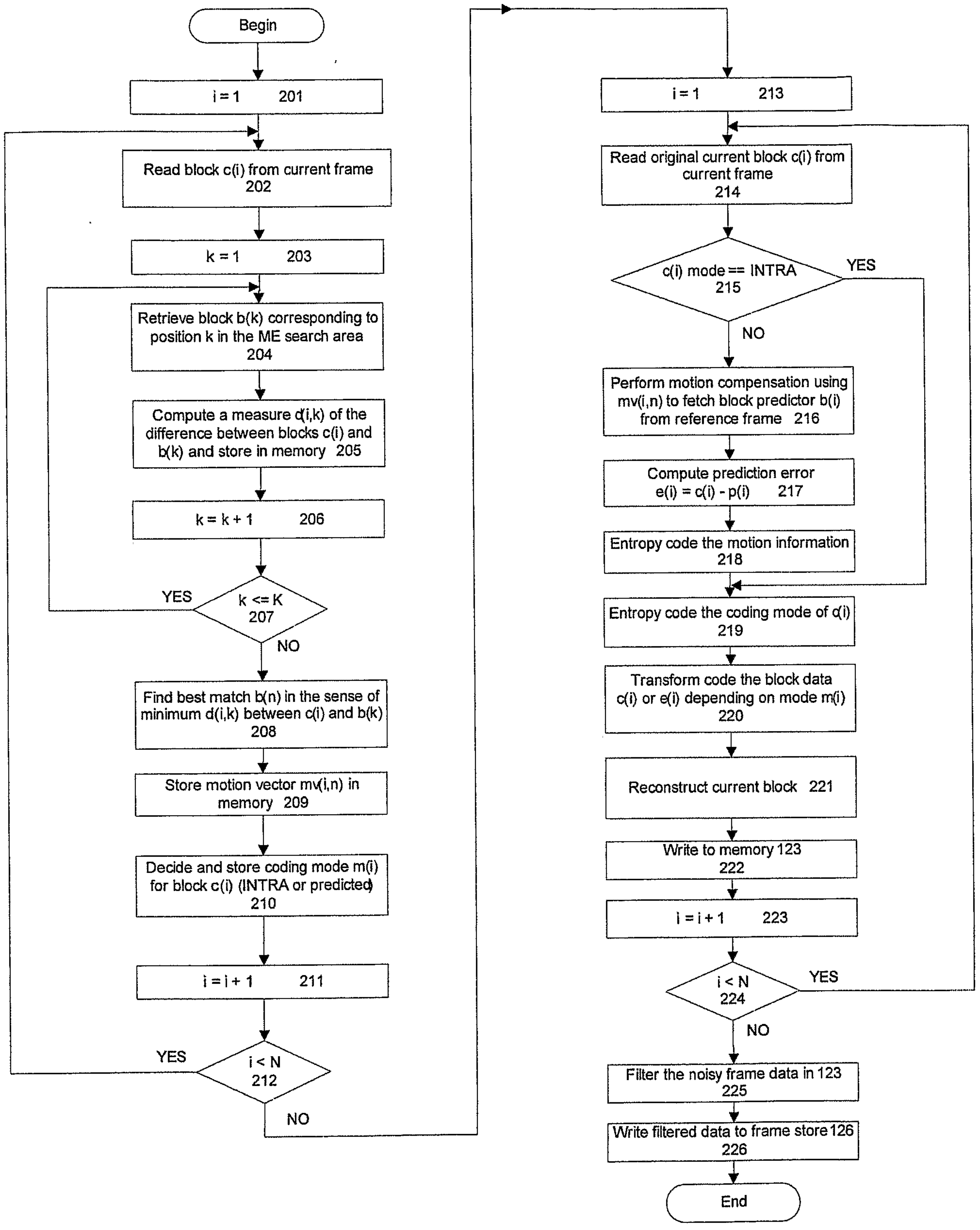
Begin
i = 1 201
Read block c(i) from current frame 202
k = 1 203
Retrieve block b(k) corresponding to position k in the ME search area 204
Compute a measure d(i,k) of the difference between blocks c(i) and b(k) and store in memory 205
k = k + 1 206
k <= K 207
YES
NO
Find best match b(n) in the sense of minimum d(i,k) between c(i) and b(k) 208
Store motion vector mv(i,n) in memory 209
Decide and store coding mode m(i) for block c(i) (INTRA or predicted) 210
i = i + 1 211
i < N 212
YES
NO
i = 1 213
Read original current block c(i) from current frame 214
c(i) mode == INTRA 215
YES
NO
Perform motion compensation using mv(i,n) to fetch block predictor b(i) from reference frame 216
Compute prediction error e(i) = c(i) - p(i) 217
Entropy code the motion information 218
Entropy code the coding mode of c(i) 219
Transform code the block data c(i) or e(i) depending on mode m(i) 220
Reconstruct current block 221
Write to memory 123 222
i = i + 1 223
i < N 224
YES
NO
Filter the noisy frame data in 123 225
Write filtered data to frame store 126 226
End

FIG. 2

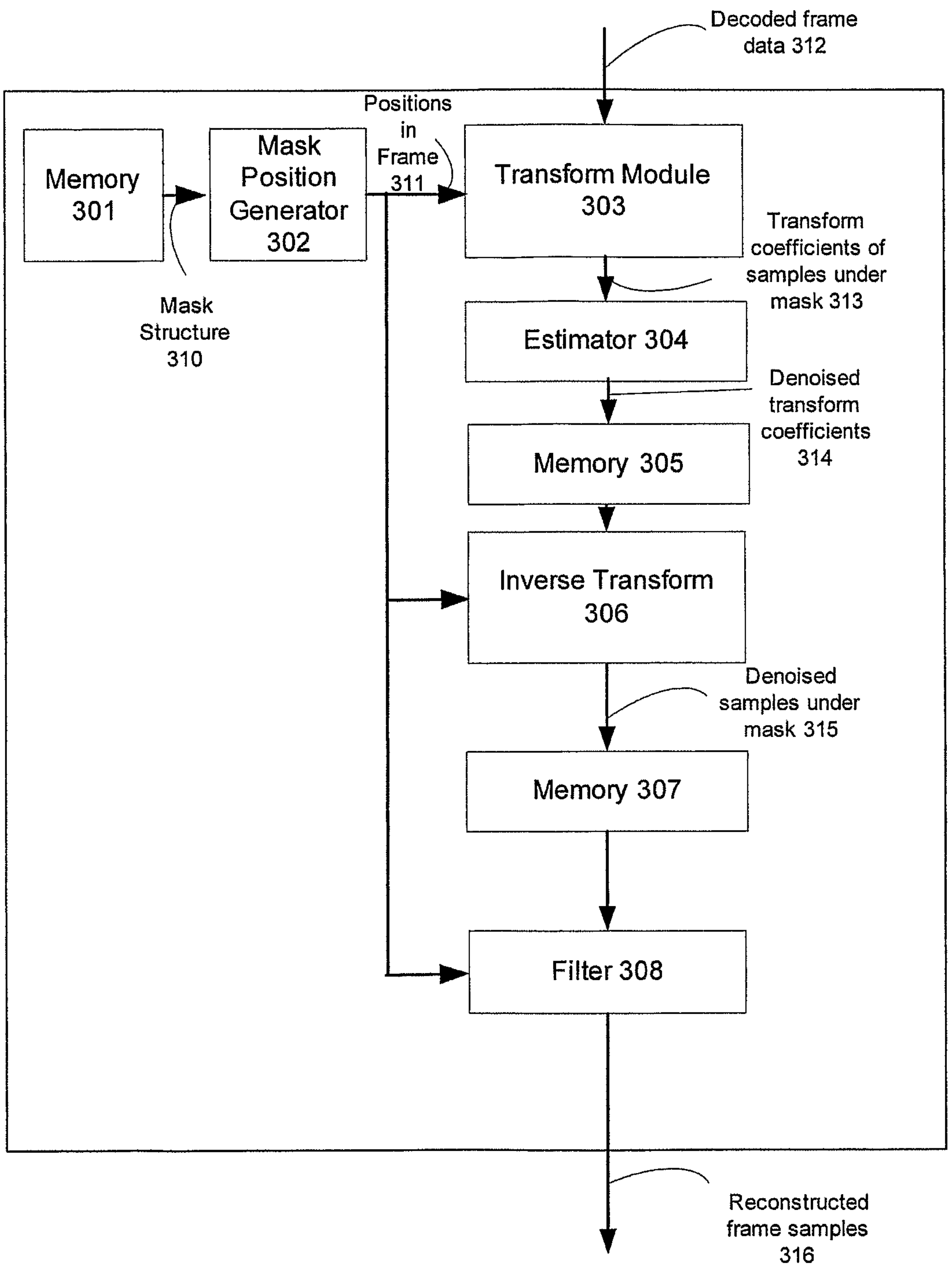
Decoded frame data 312
Memory 301
Mask Position Generator 302
Positions in Frame 311
Transform Module 303
Transform coefficients of samples under mask 313
Mask Structure 310
Estimator 304
Denoised transform coefficients 314
Memory 305
Inverse Transform 306
Denoised samples under mask 315
Memory 307
Filter 308
Reconstructed frame samples 316

FIG. 3

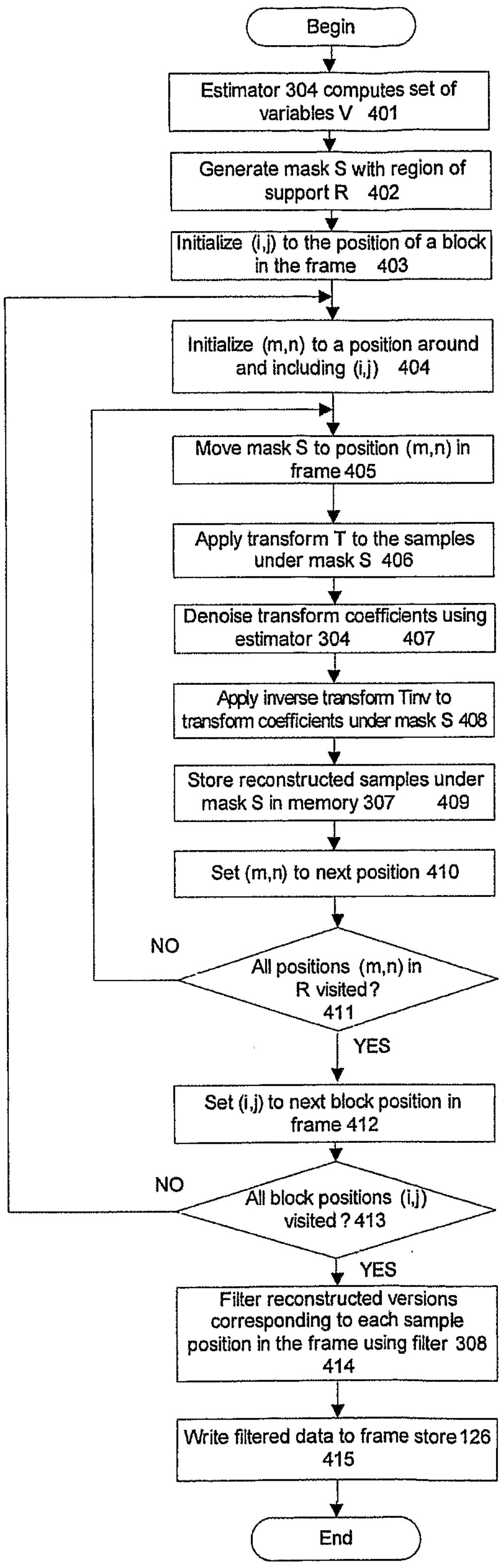
Begin
Estimator 304 computes set of variables V 401
Generate mask S with region of support R 402
Initialize (i,j) to the position of a block in the frame 403
Initialize (m,n) to a position around and including (i,j) 404
Move mask S to position (m,n) in frame 405
Apply transform T to the samples under mask S 406
Denoise transform coefficients using estimator 304 407
Apply inverse transform Tinv to transform coefficients under mask S 408
Store reconstructed samples under mask S in memory 307 409
Set (m,n) to next position 410
NO
All positions (m,n) in R visited? 411
YES
Set (i,j) to next block position in frame 412
NO
All block positions (i,j) visited? 413
YES
Filter reconstructed versions corresponding to each sample position in the frame using filter 308 414
Write filtered data to frame store 126 415
End

FIG. 4

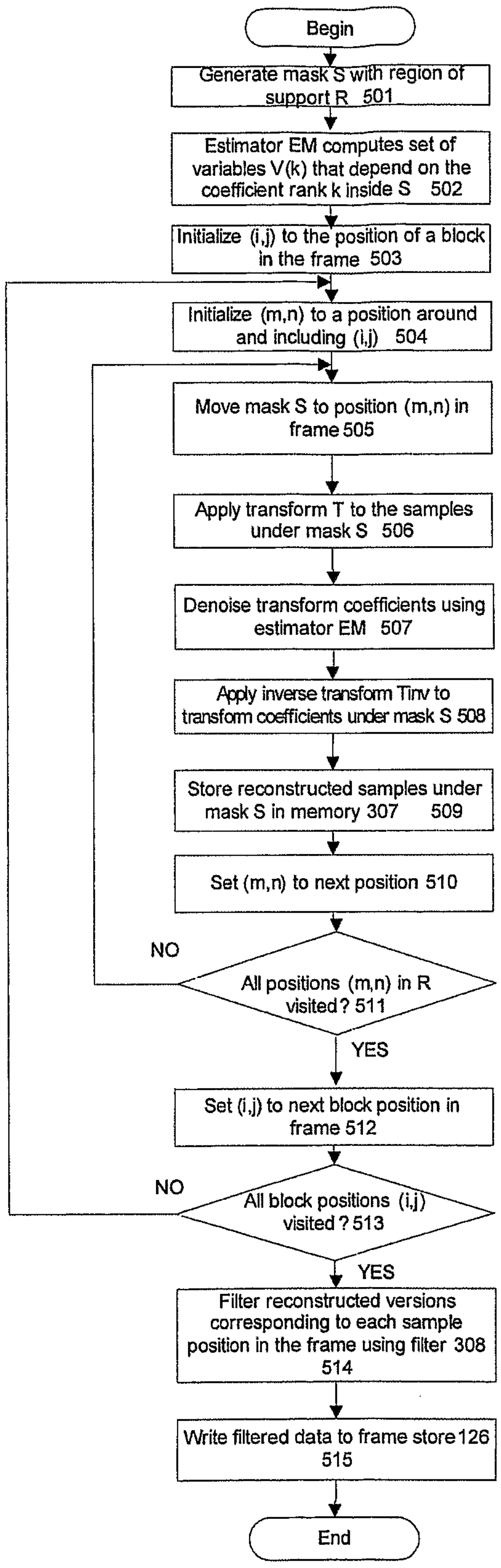
Begin
Generate mask S with region of support R 501
Estimator EM computes set of variables V(k) that depend on the coefficient rank k inside S 502
Initialize (i,j) to the position of a block in the frame 503
Initialize (m,n) to a position around and including (i,j) 504
Move mask S to position (m,n) in frame 505
Apply transform T to the samples under mask S 506
Denoise transform coefficients using estimator EM 507
Apply inverse transform Tinv to transform coefficients under mask S 508
Store reconstructed samples under mask S in memory 307 509
Set (m,n) to next position 510
NO
All positions (m,n) in R visited? 511
YES
Set (i,j) to next block position in frame 512
NO
All block positions (i,j) visited? 513
YES
Filter reconstructed versions corresponding to each sample position in the frame using filter 308 514
Write filtered data to frame store 126 515
End

FIG. 5

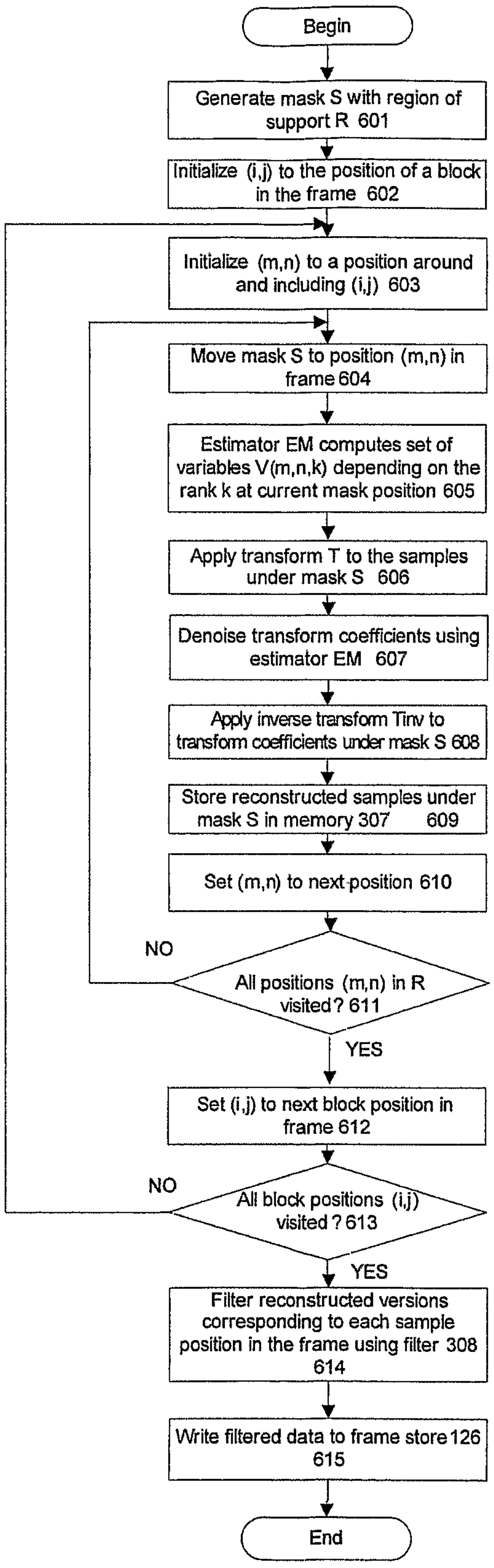
Begin
Generate mask S with region of support R 601
Initialize (i,j) to the position of a block in the frame 602
Initialize (m,n) to a position around and including (i,j) 603
Move mask S to position (m,n) in frame 604
Estimator EM computes set of variables V(m,n,k) depending on the rank k at current mask position 605
Apply transform T to the samples under mask S 606
Denoise transform coefficients using estimator EM 607
Apply inverse transform Tinv to transform coefficients under mask S 608
Store reconstructed samples under mask S in memory 307 609
Set (m,n) to next position 610
NO
All positions (m,n) in R visited? 611
YES
Set (i,j) to next block position in frame 612
NO
All block positions (i,j) visited? 613
YES
Filter reconstructed versions corresponding to each sample position in the frame using filter 308 614
Write filtered data to frame store 126 615
End

FIG. 6

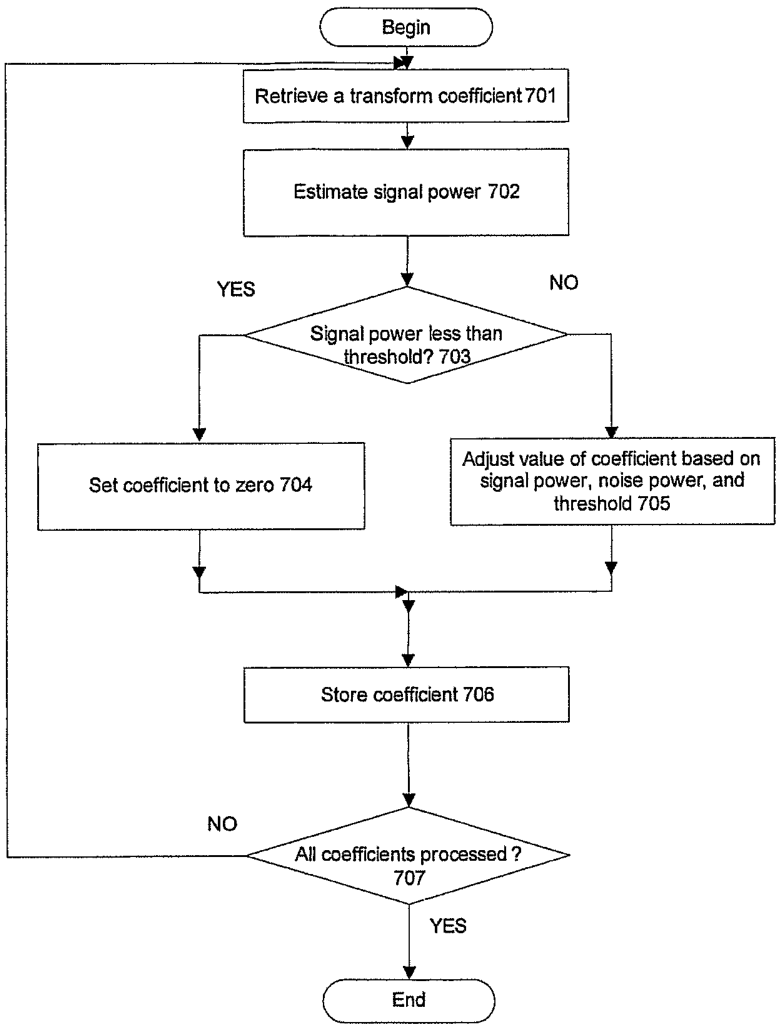
Begin
Retrieve a transform coefficient 701
Estimate signal power 702
YES
NO
Signal power less than threshold? 703
Set coefficient to zero 704
Adjust value of coefficient based on signal power, noise power, and threshold 705
Store coefficient 706
NO
All coefficients processed ? 707
YES
End

FIG. 7

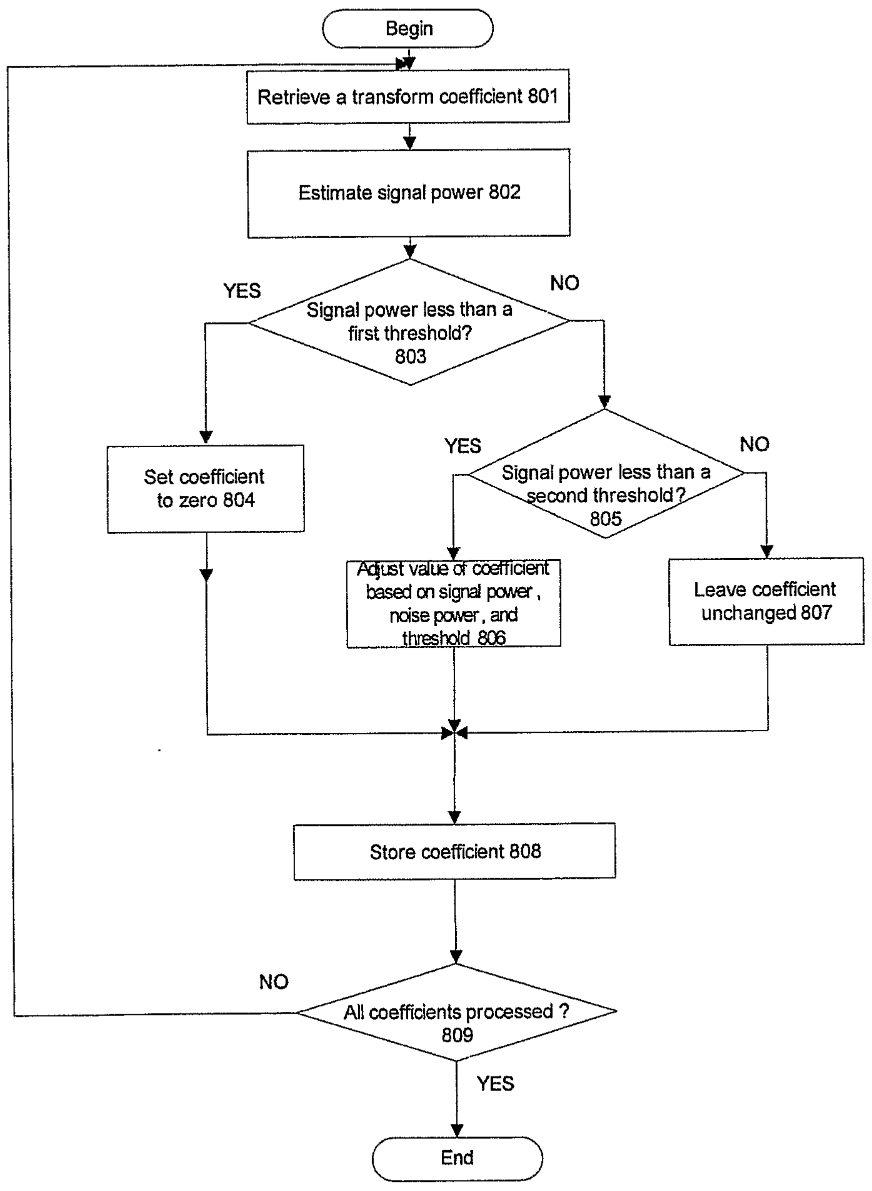
Begin
Retrieve a transform coefficient 801
Estimate signal power 802
YES
NO
Signal power less than a first threshold? 803
Set coefficient to zero 804
YES
NO
Signal power less than a second threshold? 805
Adjust value of coefficient based on signal power, noise power, and threshold 806
Leave coefficient unchanged 807
Store coefficient 808
NO
All coefficients processed ? 809
YES
End

FIG. 8

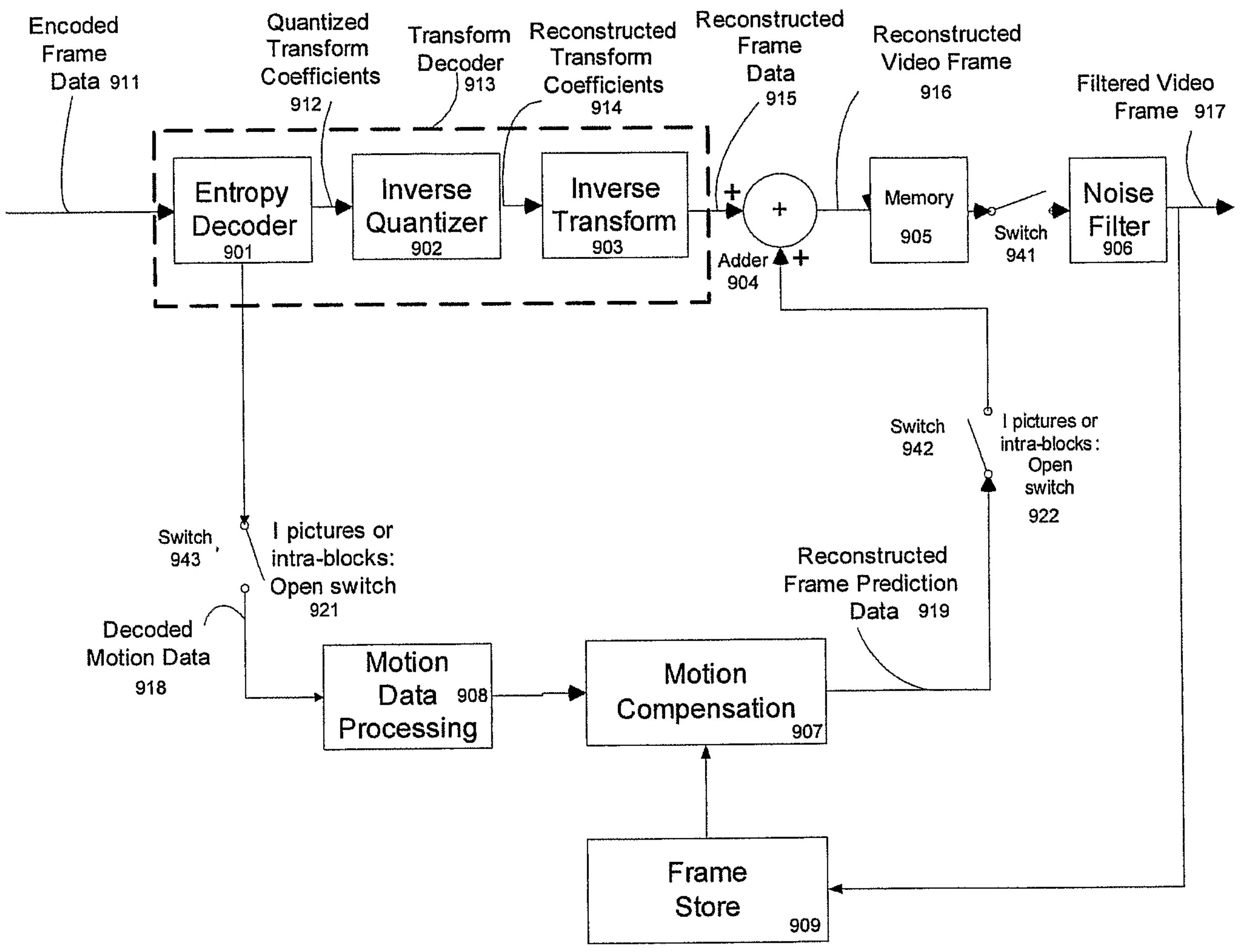
Encoded Frame Data 911
Quantized Transform Coefficients 912
Transform Decoder 913
Reconstructed Transform Coefficients 914
Reconstructed Frame Data 915
Reconstructed Video Frame 916
Filtered Video Frame 917
Entropy Decoder 901
Inverse Quantizer 902
Inverse Transform 903
+
+
Adder 904
+
Memory 905
Switch 941
Noise Filter 906
Switch 942
I pictures or intra-blocks: Open switch 922
Switch 943
I pictures or intra-blocks: Open switch 921
Decoded Motion Data 918
Reconstructed Frame Prediction Data 919
Motion Data Processing 908
Motion Compensation 907
Frame Store 909

Fig. 9

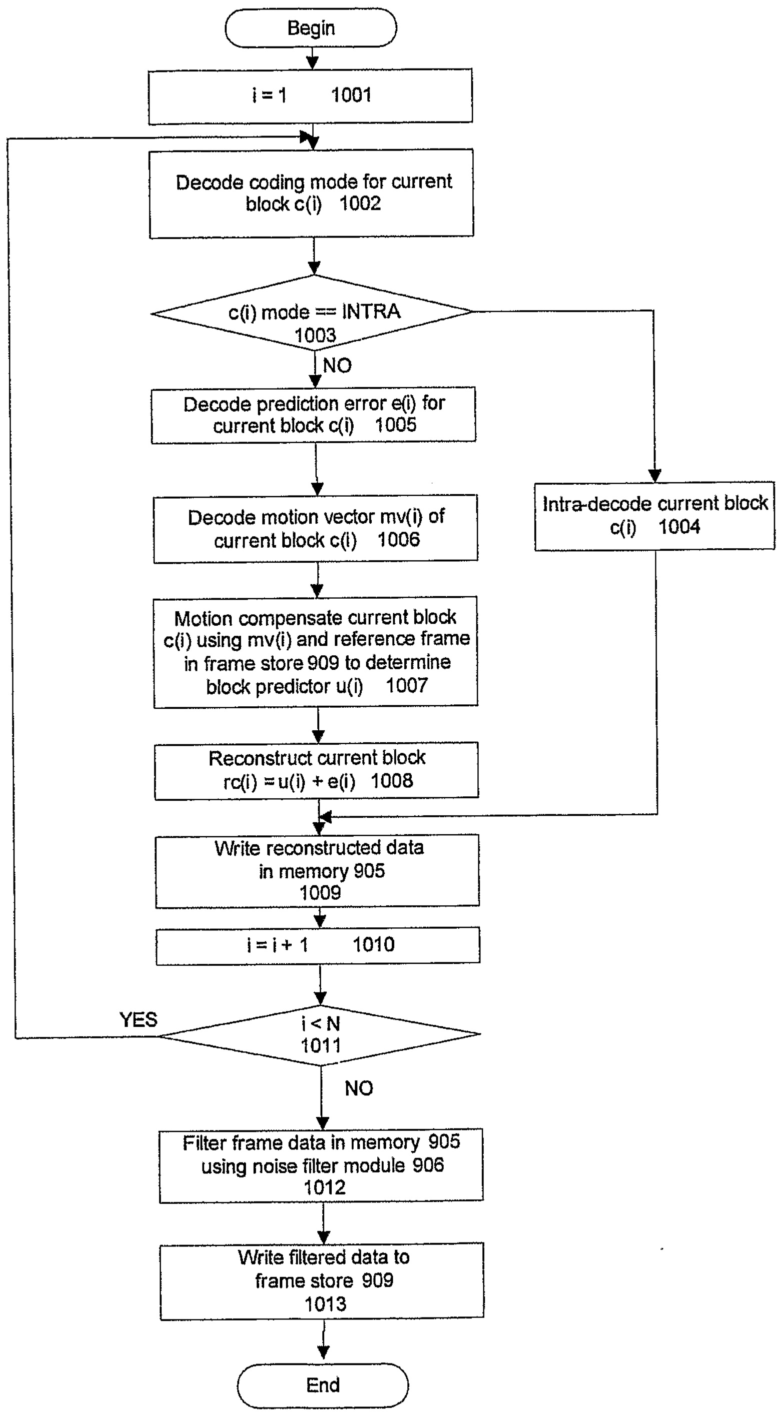
Begin
i = 1 1001
Decode coding mode for current block c(i) 1002
c(i) mode == INTRA 1003
NO
Decode prediction error e(i) for current block c(i) 1005
Decode motion vector mv(i) of current block c(i) 1006
Motion compensate current block c(i) using mv(i) and reference frame in frame store 909 to determine block predictor u(i) 1007
Reconstruct current block rc(i) = u(i) + e(i) 1008
Intra-decode current block c(i) 1004
Write reconstructed data in memory 905 1009
i = i + 1 1010
YES
i < N 1011
NO
Filter frame data in memory 905 using noise filter module 906 1012
Write filtered data to frame store 909 1013
End

FIG. 10

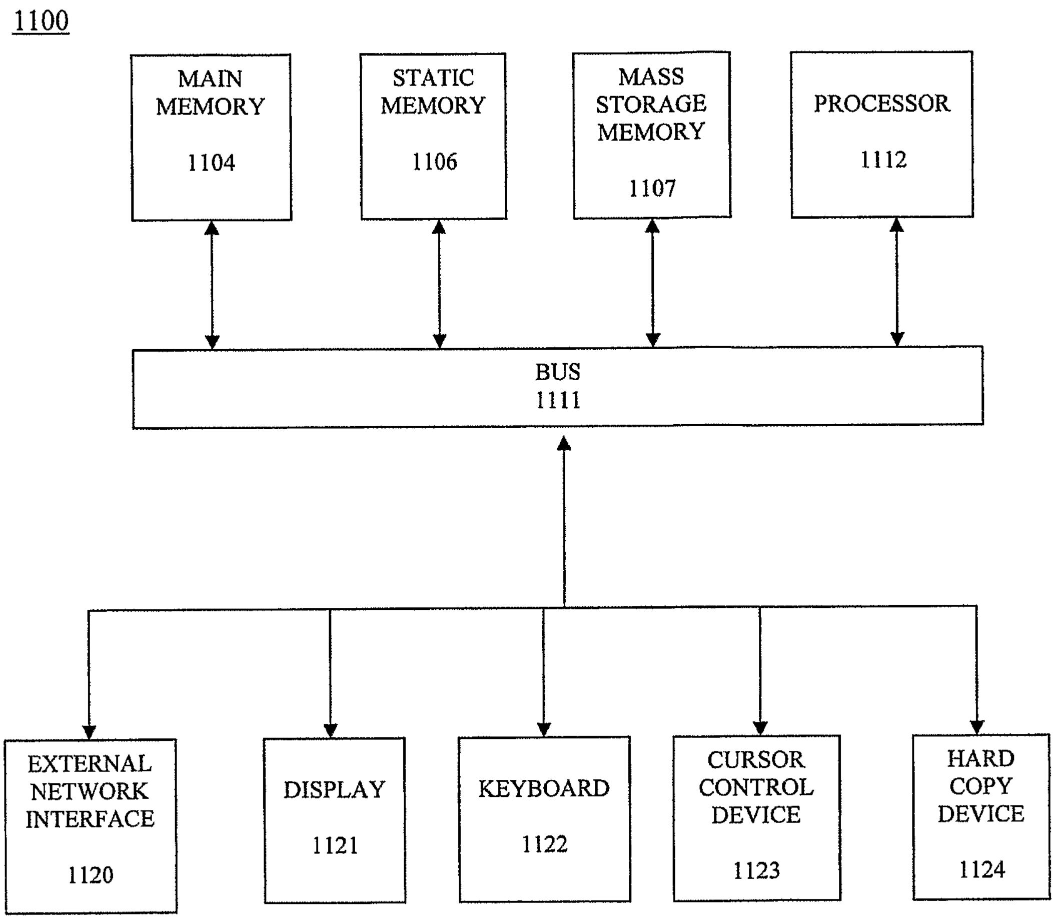
1100
MAIN MEMORY 1104
STATIC MEMORY 1106
MASS STORAGE MEMORY 1107
PROCESSOR 1112
BUS 1111
EXTERNAL NETWORK INTERFACE 1120
DISPLAY 1121
KEYBOARD 1122
CURSOR CONTROL DEVICE 1123
HARD COPY DEVICE 1124

FIG. 11